# Experimental Study on Speed-Up Techniques for Timetable Information Systems *

Reinhard Bauer, Daniel Delling, and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany,
{rbauer,delling,wagner}@ira.uka.de

**Abstract.** During the last years, impressive speed-up techniques for DIJKSTRA's algorithm have been developed. Unfortunately, recent research mainly focused on road networks. However, fast algorithms are also needed for other applications like timetable information systems. Even worse, the adaption of recently developed techniques to timetable information is more complicated than expected.
In this work, we check whether results from road networks are transferable to timetable information. To this end, we present an extensive experimental study of the most prominent speed-up techniques on different types of inputs. It turns out that recently developed techniques are much slower on graphs derived from timetable information than on road networks. In addition, we gain amazing insights into the behavior of speed-up techniques in general.

## 1 Introduction

Computing shortest paths in networks is used in many real-world applications like routing in road networks, timetable information, or air-plane scheduling. In general, DIJKSTRA's algorithm [1] can solve this problem. Unfortunately, the algorithm is too slow to be used on huge datasets, e.g. the US road network has more than 20 million nodes. In order to reduce query times for typical instances like road or railway networks, several speed-up techniques have been developed during the last years (see [2, 3] for an overview). Most recent research [4, 5] even made the calculation of the distance within a road network a matter of microseconds.

Unfortunately, due to the availability of huge road networks, recent research focused only on such networks [6]. However, fast algorithms are needed for other applications as well. One might expect that all speed-up techniques can simply be used in any other application, yet several problems arise: on the one hand, several assumptions which hold for road networks may not hold for other networks, e.g. in timetable information bidirectional search is prohibited as the arrival time is unknown in advance. Performance is the other big issue. The fastest methods [4, 5] heavily exploit properties of road networks in order to gain their huge speed-ups. Furthermore, most of the developed techniques only work in *static* scenarios, i.e. edge weights do not change between two requests. However, in railway networks, delays occur frequently. Thus, a solution for the *dynamic* timetable information problem is required.

* Partially supported by the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

ATMOS 2007 (p.209-225)
7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2007/1169

In this work, we evaluate the most prominent speed-up techniques on different types of input classes. At a glance, using the techniques on time-expanded [7] graphs for timetable information seems promising. Since road networks seem to have similar properties as railway networks—both incorporate some kind of natural hierarchy and both are sparse—one might expect that speed-up techniques yield the same performance as on road networks. However, our study reveals that speed-up techniques perform significantely worse on time-expanded graphs than on road networks. Even worse, the speed-ups obtained are below the blow-up factor of approximately 250 that exists between the time-dependent and time-expanded model [7]. As a consequence, a plain time-dependent DIJKSTRA on the time-dependent graph is faster than any speed-up techniques on the corresponding time-expanded graph. With the obtained results, we conclude that for pure performance issues the time-dependent model is somewhat superior to the time-expanded model. In addition, delays seem to be incorporated easier by the time-dependent approach.

In addition, our extensive experimental study leads to intriguing insights into the behavior of speed-up techniques. For small world inputs, the biggest speed-up is achieved by simply switching from uni- to bidirectional search and almost all speed-up techniques do *not* yield an additional speed-up. Moreover, we reveal the influence of density and diameter on the techniques. As most algorithms have only been tested on road networks, these new results are of independent interest.

### 1.1 Related Work

Systematic experiments of speed-up techniques can only be found in [8]. However, in their work, the authors only use condensed railway networks and after its publication, several additional speed-up techniques have been developed which we incorporate in this work. In [9] additional tests—besides road networks—on grid graphs are performed.

There has been some research on adapting speed-up techniques to timetable information. In [10] basic speed-up techniques are used in time-dependent and time-expanded timetable informations graphs. In [11], the multi-level speed-up technique is applied on railway graphs. Geometric containers were evaluated in [12] on such graphs as well. However, to our best knowledge, no extensive tests incorporating all recently developed speed-up techniques have been published yet.

### 1.2 Overview

This paper is organized as follows. The most prominent speed-up techniques are shortly introduced in Section 2. In Section 3 we briefly discuss existing approaches for modeling timetable information as graphs. For all three approaches we discuss advantages and disadvantages with a focus on the effort of adapting speed-up techniques to each model. Our extensive experimental study is located in Section 4, where we evaluate the speed-up techniques from Section 2 on several real-world and synthetic datasets. Our work is concluded by a summary and possible future work in Section 5.

## 2 Speed-Up Techniques

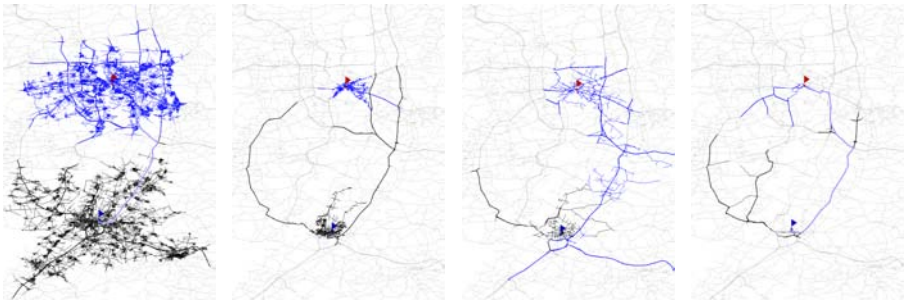
Here, we briefly present those speed-up techniques which are evaluated in Section 4 (for a more detailed overview see [2, 3]). Due to the fact that many speed-up techniques exist, we restrict ourselves to the most prominent ones and to those which do *not* need a layout of the input graph. In addition, we do not consider transit-node routing, as it was especially tuned for road networks [5]. For all techniques, we use the most sophisticated variant.

**Bidirectional DIJKSTRA.** The most straightforward speed-up technique is bidirectional search. An additional search is started from the target node and the query stops as soon as both searches meet. The tuning parameter of this approach is the way forward and backward search are alternated. We here use a strategy that strictly alternates between both searches, balancing the work between them. Note that most sophisticated methods are bidirectional approaches.

**ALT [13].** Goal directed search, also called $A^*$ [14], pushes the search towards a target by adding a potential to the priority of each node. Given a 2-dimensional layout, the usage of Euclidean potentials requires no preprocessing. The ALT algorithm, introduced in [13], obtains the potential from the distances to certain landmarks in the graph. Although this approach requires a preprocessing step, it is superior with respect to search space and query times. In this work, we use the latest variant of ALT, introduced in [15], with 16 *maxCover* landmarks as representative of goal-directed search. The main advantages of ALT is its simple implementation and it can be used—without modification for most updates—in a *dynamic* and *time-dependent* scenario [16], i.e. edge weights may change between two queries. The main downside of ALT are very fluctuating query times.

**Arc-Flags [17, 18].** This approach uses a *pruning* strategy, i.e. by attaching additional data to edges, a modified DIJKSTRA checks whether an edge can or cannot be on the shortest path to the target. More precisely, the Arc-Flag approach partitions the graph into cells and attaches a label to each edge. A label contains a flag for each cell indicating whether a shortest path to the corresponding cell exists that starts with this edge. As a result, Arc-Flag DIJKSTRA often *only* visits those edges which lie on the shortest path of a long-range query. However, no speed-up can be achieved for queries within a cell and the effort of the preprocessing is very high. In this work, we use the variant as described in [19].

**Highway Hierarchies [20].** This approach is a purely hierarchical method, i.e. an approach trying to exploit the hierarchy of a graph. Therefore, the network is contracted and then "important" edges—the highway edges—are identified. By rerunning those two steps, a natural hierarchy of the network is obtained. The contraction phase builds the *core* of a level and adds shortcuts to the graph. The identification of highway edges



(a) Dijkstra (b) ALT (c) REAL (d) Arc-Flags

**Fig. 1.** Search Space of some of the examined (bidirectional) speed-up techniques.

is done by local DIJKSTRA executions. In this work, we use the variant of Highway Hierarchies (HH) as described in [20]. This variant stops building the hierarchy at a certain point and computes a *distance* table containing all distances between the core-nodes of the highest level. The advantages of HH are very low preprocessing and query times (15 minutes of preprocessing on the Western European road network result in query times of 0.5 ms). However, this approach loses performance when using other metrics than travel times [21].

**RE/REAL [9].** *Reach* [22] is a centrality measure based on the intuition that a node is important, if it is situated in the middle of long shortest paths. In [22], reach is used as *node-label* in order to prune the search. Some crucial disadvantages, e.g. preprocessing time, are remedied by enriching the graph by shortcuts in [9]. In addition, this approach naturally combines with ALT yielding impressive speed-ups in road networks. The RE algorithm is a bidirectional reach-pruning DIJKSTRA on a shortcut-enriched graph, while REAL is the combination of RE and ALT. Note that RE can be interpreted as a hierarchical method. RE has similar advantages and disadvantages like HH, but preprocessing takes longer than for HH. The advantage of RE over HH is its sound combination with ALT, which cannot be combined with HH easily [21].

**Example.** Figure 1 shows the search space of some of the above mentioned speed-up techniques running the same query on the German road network. More precisely, the source of the query is the university of Karlsruhe, the target the university of Mannheim. A black edge depicts that it has been *relaxed* by the forward search, blue edges show the backward search. Note that for REAL, shortcuts are inserted into the graph which we unpack for visualization. As a consequence, the search space may look bigger than for other techniques, but the number of settled nodes may be smaller.

We observe that ALT gives the search an excellent sense of goal-direction but almost all nodes are visited near source and target of the query. By adding reach to ALT this drawback is compensated by pruning unimportant nodes. The search space of Arc-Flags seems to be only slightly bigger than the actual shortest path.

## 3 Modeling Timetable Information

In this section, we briefly present existing approaches to model (dynamic) timetable information as graphs (cf. [7] for details). In addition, we discuss problems of adapting speed-up techniques to these models and how well delays can be covered.

**Condensed Model.** The easiest model is the *condensed* model. Here, a node is introduced for each station and an edge is inserted iff a direct connection between two stations exist. The edge weight is set to be the minimum travel time over all possible connections between these two stations. The advantage of this model is that the resulting graphs are small and we are able to use speed-up techniques without modification. Unfortunately, several drawbacks exist. First of all, this model does not incorporate the actual departure time from a given station. Even worse, travel times highly depend on the time of the day and the time needed for changing trains is also not covered by this approach. As a result, the calculated travel time between two arbitrary stations in such a graph is only a *lower bound* of the real travel time. Furthermore, delays can hardly be incorporated by this model.

**Fig. 2.** Condensed network of the European timetable information data, provided by HaCon [23] for scientific use.

**Time-Dependent Model.** This model tries to remedy the main disadvantages of the condensed model. The main idea is to use *time-dependent* edges. Hence, each station is also modeled by a single node and an edge is again inserted iff a direct connection between two stations exist. But unlike for the condensed model, several weights are assigned to each edge. Each weight represents the travel time of a train running from one station to another. The edge used during a query is then picked according to the departure time from the station. See Fig. 3 for a small example. The advantage of this model is its still small size and the obtained travel time is feasible. Furthermore, delays can easily be incorporated: the corresponding weight—representing the delayed connection—of an edge can simply be increased. However, adapting speed-up techniques to time-dependent graphs is more complicated than expected. While for time-independent graphs speed-ups of over one million can be achieved [5], best results for time-dependent graphs only yield speed-ups of factor 5 [16]. In addition, this model does not cover transfer times, yet this can be remedied as shown in [24]. Note that the time-dependent model can be interpreted as an extension of the condensed model. In this work we evaluate speed-up techniques on the condensed model in order to select techniques that are worth adapting to the dynamic time-dependent model.

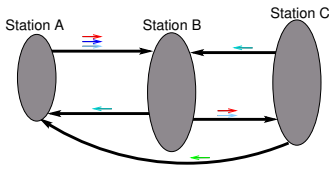



**Fig. 3.** Time-dependent model.

**Time-Expanded Model.** This model does not rely on time-dependent edge weights and thus it is much easier to use existing speed-up techniques in this model. Here, a node is used for each arrival and departure *event*. An edge is inserted for each connection between two events. Figure 4 gives an example. The main downside of this approach is that the resulting graphs are much bigger than for the time-dependent approach. For our datasets, the number of nodes is roughly 250 times higher. Note that such graphs are strongly connected as timetables are periodic.

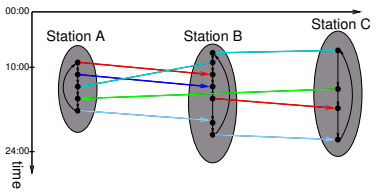



**Fig. 4.** Time-expanded model.

In general, most unidirectional speed-up techniques can be used out-of-the-box on such a time-expanded graph. However, sophisticated methods gain their speed-ups from bidirectional search that needs to know the exact target node. Even worse, RE and HH only work correctly if used in a bidirectional manner. Unfortunately, in this model each node represents a specific event within the network and thus it is complicated to pick the target node from which to start the backward search. In addition, some unidirectional approaches, e.g. unidirectional ALT, also need the exact target node in order to work properly. Another pitfall originates from the model. The ordering of nodes within a station is very important for the correctness of timetable information queries. Whenever a delay occurs, trains may arrive in a different order than expected, leading to a complete change of the inner-edge structure of a station. As a consequence, delays yield changes in the topology within the network which results in a bigger effort of updating the preprocessed data of the speed-up techniques. Thus, adapting techniques to a dynamic time-expanded model appears to be very complicated.

Note that transfer times are not covered correctly. For this reason, this model is called the *simple* time-expanded model. However, this can be remedied by an *extended* model, but the graph size additionally increases by a factor of approximately 2. In this work, we evaluate the speed-up techniques on the static simple time-expanded model in order to pick the most promising technique that is worth adapting to the dynamic extended time-expanded model.

## 4 Experiments

In this section, we present an extensive experimental evaluation of the speed-up techniques on different types of graphs. Our implementation is written in C++ using solely the STL. As priority queue we use a binary heap. Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.1. The machine is clocked at 2.6 GHz, has 16 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.1, using optimization level 3.

*Default Settings.* Unlike otherwise stated, we use the following settings. For ALT, we use 16 *maxCover* landmarks. In our Arc-Flag setup, we use 128 cells obtained from METIS [25]. In addition, we evaluate the hierarchical RE algorithm [9] and Highway Hierarchies (HH) [20]. The performance of both approaches highly depends on the chosen preprocessing parameters which we here tune manually. For HH, we use a distance

table as soon as the contracted graph has less than 10 000 nodes. Moreover, we evaluate the combination of RE and ALT, named REAL, *without* reach-aware landmarks [26].

Unless otherwise stated, we determine the query-performance of all algorithms by running 10 000 random queries. We log the average execution time and number of settled nodes of the queries. By settled nodes we denote the number of nodes taken from the priority queues.

### 4.1 Timetable Information

*Condensed Model.* We start our experimental study with the condensed network of Europe, based on timetable information data provided by HaCon [23] for scientific use. The graph has 29 578 nodes and 86 566 edges. In order to check whether speed-ups derive from the topology of the network or if they are due to the used metric we use—besides travel times—three additional metrics: *distance* depicts the real distance between two stations, *unit* assigns weight 1 to each edge, and *random* reassigns each edge weight with a value between 1 and 1000 picked uniformly at random. The resulting figures are shown in Tab. 1.

We observe that plain DIJKSTRA settles the same number of nodes independent of the applied metric. However, query times vary: DIJKSTRA is two times faster on the distance metric than on the random one. The number of DECREASEKEY operations causes these different running times. Surprisingly, switching to bidirectional DIJKSTRA has a completely different impact for different metrics. While for travel times and distances, a speed-up of factor 2 is observed, queries using the unit metric get 12 times faster. We observe several direct connections within the network. Thus, setting the weight of these edges to 1 drastically reduces search space of bidirectional DIJKSTRA as forward

**Table 1.** Performance of speed-up techniques on the condensed railway network of Europe. Figures are based on 10 000 random queries. *Prepro* shows the computation time of the preprocessing in *minutes* and the eventual *additional* bytes per node needed for the preprocessed data. For queries, the search space is given in number of settled nodes, execution times are given in milliseconds. Due to the graph size, we use the distance table for HH as soon as the core has less than 1 000 nodes.

| | travel times | | | | distance | | | | unit | | | | random | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY |
| | min | B/n | #sett. | ms | min | B/n | #sett. | ms | min | B/n | #sett. | ms | min | B/n | #sett. |
| Dijkstra | 0.0 | 0 | 14761 | 3.48 | 0.0 | 0 | 14603 | 2.82 | 0.0 | 0 | 14691 | 3.35 | 0.0 | 0 | 14549 |
| BiDijkstra | 0.0 | 0 | 7520 | 1.83 | 0.0 | 0 | 8615 | 1.69 | 0.0 | 0 | 1158 | 0.27 | 0.0 | 0 | 1515 |
| uni ALT | 0.1 | 128 | 1191 | 0.47 | 0.1 | 128 | 1007 | 0.37 | 0.1 | 128 | 1840 | 0.90 | 0.1 | 128 | 1835 |
| ALT | 0.1 | 128 | 348 | 0.21 | 0.1 | 128 | 374 | 0.21 | 0.1 | 128 | 109 | 0.10 | 0.1 | 128 | 108 |
| uni Arc-F. | 0.6 | 47 | 236 | 0.13 | 0.5 | 47 | 327 | 0.14 | 0.6 | 47 | 160 | 0.08 | 0.7 | 47 | 178 |
| Arc-Flags | 1.1 | 94 | 50 | 0.03 | 1.0 | 94 | 75 | 0.03 | 1.1 | 94 | 19 | 0.01 | 1.5 | 94 | 26 |
| RE | 0.1 | 27 | 272 | 0.13 | 0.1 | 20 | 258 | 0.12 | 0.1 | 16 | 377 | 0.15 | 0.8 | 22 | 739 |
| uni REAL | 0.2 | 155 | 116 | 0.12 | 0.2 | 148 | 87 | 0.09 | 0.2 | 144 | 687 | 0.64 | 0.9 | 150 | 751 |
| REAL | 0.2 | 155 | 72 | 0.08 | 0.2 | 148 | 70 | 0.07 | 0.2 | 144 | 66 | 0.09 | 0.9 | 150 | 81 |
| HH | 0.1 | 46 | 88 | 0.04 | 0.1 | 78 | 226 | 0.11 | 0.1 | 24 | 338 | 0.12 | 0.1 | 38 | 125 |

and backward search meet earlier. This observation also holds somewhat weaker for the random metric, here the speed-up is of factor 10.

Analyzing our speed-up techniques, all approaches are able to preprocess the graph in less than 1 minute. The fastest technique is bidirectional Arc-Flags having query times of below 30 $\mu$s for all metrics. As for bidirectional DIJKSTRA, the lowest query times are achieved for the unit metrics which is again due to direct connections. RE requires the lowest amount of additional memory and thus has the best combination of query times and preprocessing. Nevertheless, as we use the condensed model, the obtained travel times cannot be used in a real world environment (cf. Section 3).

*Time-Expanded Model.* Our second set of experiments is executed on three simple time-expanded graphs (cf. Section 3). The first shows the local traffic of Berlin/Brandenburg, has 2 599 953 nodes and 3 899 807 edges, the second one represents local traffic of the Ruhrgebiet (2 277 812 nodes, 3 416 597 edges), and the last graph depicts long distance connections of Europe (1 192 736 nodes, 1 789 088 edges). Table 2 gives an overview of the performance of speed-up techniques on these instances.

Note that RE, ALT, and HH cannot be used out-of-the-box for time-expanded networks (cf. Section 3). In order to gain insights in the performance of these techniques, we also use bidirectional speed-up techniques by picking a random event at the target station. Thus, these bidirectional experiments are intended to give hints whether it is worth focusing on adapting bidirectional search to such graphs. Only unidirectional Arc-Flags—with a partitioning by station—are applicable, which perform roughly 12-18 times faster than unidirectional DIJKSTRA. But when switching to bidirectional search we gain another speed-up of factor 6-10. Thus, it may be worth focusing on the question how to use bidirectional search in this scenario. However, we observe very long preprocessing times for Arc-Flags on these networks. Although other approaches have smaller search space, e.g. REAL, the smaller computational overhead of Arc-Flags yields smaller query times. However, only ALT and HH can preprocess all graphs in below one hour. RE seems to have problems on the local traffic networks as preprocessing takes longer than 3 hours and speed-ups are only mild, while this does not hold

**Table 2.** Performance of speed-up techniques on time-expanded railway networks.

| | Berlin/Brandenburg | | | | Ruhrgebiet | | | | long distance | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | |
| | min | B/n | #sett. | ms | min | B/n | #sett. | ms | min | B/n | #sett. | ms |
| Dijkstra | 0 | 0 | 1299830 | 406.2 | 0 | 0 | 1134420 | 389.2 | 0 | 0 | 609352 | 221.2 |
| BiDijkstra | 0 | 0 | 496281 | 151.3 | 0 | 0 | 389577 | 122.8 | 0 | 0 | 143613 | 43.8 |
| uni ALT | 10 | 128 | 383921 | 133.6 | 10 | 128 | 171760 | 64.7 | 5 | 128 | 71194 | 26.0 |
| ALT | 10 | 128 | 47764 | 22.9 | 10 | 128 | 59516 | 30.5 | 5 | 128 | 31367 | 15.0 |
| uni Arc-F. | 2240 | 24 | 172362 | 72.2 | 2323 | 24 | 158174 | 66.4 | 1008 | 24 | 74737 | 32.4 |
| Arc-Flags | 4479 | 48 | 24004 | 9.2 | 4646 | 48 | 28448 | 10.7 | 2016 | 48 | 10560 | 3.5 |
| RE | 182 | 39 | 27095 | 25.5 | 290 | 45 | 38397 | 39.8 | 63 | 43 | 8978 | 8.3 |
| uni REAL | 192 | 167 | 20062 | 22.2 | 300 | 173 | 16649 | 21.1 | 68 | 171 | 6335 | 8.8 |
| REAL | 192 | 167 | 4159 | 6.6 | 300 | 173 | 7867 | 13.3 | 68 | 171 | 2479 | 4.5 |
| HH | 38 | 263 | 5285 | 56.1 | 65 | 202 | 9528 | 196.2 | 12 | 386 | 1930 | 7.3 |

for long distance connections. Regarding query times, HH has also problems with both local traffic networks: on Berlin/Brandenburg, HH is only 3 times faster than bidirectional DIJKSTRA, and on the Ruhrgebiet, HH is even slower. The problems of RE/HH derive from a weaker hierarchy within the local networks compared to the long-distance graph. Local traffic networks do not incorporate high-speed trains while the latter do.

Summarizing, the fastest techniques yield only mild speed-ups of a factor below 80. And this speed-up can only be achieved when using bidirectional search. As a consequence, the blow-up of time-expanded graphs of factor 250 over the condensed—and hence also time-dependent—graphs cannot be compensated. Plain DIJKSTRA on a corresponding condensed network would be faster—with respect to query times—than any other speed-up technique on the time-expanded model. Note that our input from Tab. 1 covers even more stations than any input from Tab. 2. Also note that plain DIJKSTRA can be used in a dynamic time-dependent scenario [27], and time-dependent ALT achieves an additional speed-up of factor 5 over plain DIJKSTRA [16].

### 4.2 Road Networks

Like railway networks, road graphs incorporate some kind of hierarchy. Hence, one might expect that speed-up techniques have similar performance on those two types of networks. We evaluate the German road network, provided by PTV AG [28] for scientific use. It has 4 377 307 nodes and 10 667 837 edges. We use three different metrics: travel times, distance, and random. The latter reassigns edge weights uniformly at random from 1 to 1000 to each edge. We hereby want to test whether the speed-up techniques rely on the topology of the network or the speed-up derive from the used metric. The results can be found in Tab. 3.

As expected, plain DIJKSTRA settles the same number of nodes for each metric. Stunningly, query times vary heavily when switching metrics: DIJKSTRA's algorithm is two times faster on the distance metric than on the random. This derives from the number of DECREASEKEY operations of the used priority queue. However, when switching from uni- to bidirectional DIJKSTRA, the situation changes. Surprisingly, the number

**Table 3.** Performance of speed-up techniques on the German road graph using different metrics.

| | travel times | | | | distance | | | | random | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | |
| | min | B/n | #settled | ms | min | B/n | #settled | ms | min | B/n | #settled | ms |
| Dijkstra | 0 | 0 | 2214820 | 1078.2 | 0 | 0 | 2159310 | 625.8 | 0 | 0 | 2256530 | 1335.4 |
| BiDijkstra | 0 | 0 | 1210570 | 545.0 | 0 | 0 | 1428140 | 405.7 | 0 | 0 | 1006260 | 530.0 |
| uni ALT | 23 | 128 | 139121 | 51.2 | 18 | 128 | 95385 | 33.823 | 23 | 128 | 143551 | 59.4 |
| ALT | 23 | 128 | 22150 | 12.4 | 18 | 128 | 45496 | 23.1 | 23 | 128 | 21433 | 12.2 |
| uni Arc-F. | 976 | 39 | 24290 | 10.6 | 720 | 39 | 59094 | 24.2 | 1139 | 39 | 24509 | 14.0 |
| Arc-Flags | 1952 | 78 | 1092 | 0.5 | 1440 | 78 | 13038 | 5.4 | 2278 | 78 | 897 | 0.4 |
| RE | 18 | 22 | 5080 | 3.1 | 20 | 27 | 10666 | 9.4 | 20 | 30 | 4879 | 3.5 |
| uni REAL | 41 | 150 | 1804 | 1.8 | 38 | 155 | 1642 | 2.1 | 43 | 158 | 2369 | 2.7 |
| REAL | 41 | 150 | 1035 | 1.2 | 38 | 155 | 1556 | 2.343 | 43 | 158 | 1130 | 1.4 |
| HH | 4 | 99 | 682 | 0.5 | 9 | 122 | 3602 | 3.8 | 5 | 83 | 1039 | 0.9 |

of settled nodes is *not* the same for each metric. The reason for this are the motorways which are favored differently by each metric.

Analyzing the speed-up techniques, we observe very high preprocessing times for Arc-Flags which is due to the high number of DIJKSTRA executions during preprocessing. However, Arc-Flags yields the fastest query times although the search space is higher than for HH which is due to a smaller number of additional operations for Arc-Flags, yet HH can preprocess the complete German network much faster than any other technique. This result is not very surprising since HH was tuned for road networks and exploits properties of the (European) datasets. For example, curves on motorways are often modeled by a path with many degree-2 nodes which are shortcut during the preprocessing of HH. The same holds for RE. For ALT, we observe that the number of settled nodes is almost the same for travel times, unit, and random. This holds for the uni- and bidirectional variant. However, for distance the situation is different: The unidirectional variant is faster on this metric while the bidirectional is slower. As a consequence, REAL (the combination of RE and ALT) has a surprising performance on this metric. The undirectional variant is faster than the bidirectional one.

Summarizing, the distance metric seems to be very different from the other metrics. For the latter, Arc-Flags yield best query performances on road networks but for the price of high preprocessing times. HH seem to have the best trade-off between perprocessing time and query performance. But for distance, *unidirectional* REAL outperforms all other techniques.

*Similarity to Railway Networks.* Comparing Tabs. 2 and 3 we observe different performance of speed-up techniques on time-expanded graphs and road networks. So, at least for the time-expanded model the assumption of similar properties seems *not* to hold. However, comparing Tabs. 1 and 3, and taking the difference in size into account, it seems as if road networks can be used as alternative for condensed railway networks. But as graph sizes are very different from each other, we perform another test on a road network of similar size like the European railway network. We choose the road network

**Table 4.** Performance of speed-up techniques on the Luxemburg road network.

| | travel times | | | | distance | | | | unit | | | | random | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY |
| | min | B/n | #sett. | ms | min | B/n | #sett. | ms | min | B/n | #sett. | ms | min | B/n | #sett. |
| Dijkstra | 0.0 | 0 | 15293 | 3.12 | 0.0 | 0 | 15230 | 2.87 | 0.0 | 0 | 15441 | 2.69 | 0.0 | 0 | 15156 |
| BiDijkstra | 0.0 | 0 | 7691 | 1.63 | 0.0 | 0 | 9526 | 1.77 | 0.0 | 0 | 7304 | 1.28 | 0.0 | 0 | 7056 |
| uni ALT | 0.1 | 128 | 1375 | 0.53 | 0.1 | 128 | 1052 | 0.37 | 0.1 | 128 | 1099 | 0.41 | 0.1 | 128 | 1122 |
| ALT | 0.1 | 128 | 448 | 0.21 | 0.1 | 128 | 451 | 0.21 | 0.1 | 128 | 458 | 0.21 | 0.1 | 128 | 456 |
| uni Arc-F. | 0.3 | 37 | 470 | 0.17 | 0.3 | 37 | 614 | 0.23 | 0.3 | 37 | 421 | 0.15 | 0.4 | 37 | 435 |
| Arc-Flags | 0.7 | 74 | 178 | 0.06 | 0.6 | 74 | 250 | 0.09 | 0.6 | 74 | 133 | 0.05 | 0.8 | 74 | 144 |
| RE | 0.1 | 28 | 532 | 0.21 | 0.1 | 29 | 348 | 0.16 | 0.1 | 22 | 358 | 0.12 | 0.1 | 34 | 385 |
| uni REAL | 0.2 | 156 | 229 | 0.20 | 0.2 | 157 | 105 | 0.10 | 0.2 | 150 | 171 | 0.14 | 0.2 | 162 | 174 |
| REAL | 0.2 | 156 | 119 | 0.11 | 0.2 | 157 | 86 | 0.09 | 0.2 | 150 | 97 | 0.08 | 0.2 | 162 | 101 |
| HH | 0.1 | 219 | 91 | 0.05 | 0.1 | 140 | 241 | 0.12 | 0.1 | 69 | 299 | 0.14 | 0.1 | 204 | 111 |

of Luxemburg which has nodes 30 746 and 71 655 edges. Again, we use the four metrics travel times, distance, unit and random. The resulting figures can be found in Tab. 4.

We observe that for the most important—at least in our application—metric, i.e. travel times, all speed-up techniques perform very similar as on the condensed railway network. Differences in the unit and random metrics derive from direct connections within the railway network that do not exist in road networks. We conclude that road networks can be used as alternative data for the condensed model if timetable data is lacking.

*Important Subgraphs.* The European road networks include roads which are closed to public traffic, e.g. pedestrian zones, etc. By removing these roads from the German network, the number of nodes decreases to 3 523 370 and the number of edges to 8 133 531, respectively. As these roads seem unimportant to shortest path computation, one might expect that the performance of the evaluated speed-up techniques hardly changes whether they are included or not. In addition, degree-1 and degree-2 nodes seem to be unimportant for shortest paths as well: Nodes with degree 1 can only be starting or ending points of a route and degree 2 nodes can often be *shortcut*. Table 5 shows the results of all speed-up techniques if non-public roads are excluded, using the 2-core as input (3 183 701 nodes, 8 280 625 edges), the graph with shortcut degree-2 nodes (3 723 319 nodes, 9 363 584 edges), and the 2-core with shortcut degree-2 nodes (1 828 995 nodes, 5 469 750 edges). As metric, we use travel times.

Comparing the results from Tabs. 3 and 5, we observe that the search space of uni- and bidirectional DIJKSTRA decreases with the size of the subgraphs. Astonishingly, this does not hold for query times: shortcutting degree-2 nodes yields higher query times than using the 2-core. The reason for this is that the number of edges differ: the 2-core has less edges than the other subgraph. However, this fact has no influence on bidirectional ALT. The algorithm has the same performance on the first three subgraphs and surprisingly, the performance is almost the same as on the full graph. Only when using the shortcut 2-core search spaces decrease which is due to graph size.

**Table 5.** Performance of speed-up techniques on different subgraphs ofthe German road graph.

| | only public | | | no deg. 2 | | | 2-core | | | 2-core + no deg. 2 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | PREPRO | | QUERY | PREPRO | | QUERY | PREPRO | | QUERY |
| | min | B/n | #settled | min | B/n | #settled | min | B/n | #settled | min | B/n | #settled |
| Dijkstra | 0 | 0 | 1 729 390 | 0 | 0 | 1 809 350 | 0 | 0 | 1 580 610 | 0 | 0 | 913 476 |
| BiDijkstra | 0 | 0 | 974 453 | 0 | 0 | 978 311 | 0 | 0 | 855 943 | 0 | 0 | 497 760 |
| uni ALT | 14 | 128 | 112 814 | 17 | 128 | 119 778 | 14 | 128 | 106 668 | 8 | 128 | 59 907 |
| ALT | 14 | 128 | 21 914 | 17 | 128 | 19 589 | 14 | 128 | 19 757 | 8 | 128 | 10 668 |
| uni Arc-F. | 610 | 37 | 20 583 | 794 | 40 | 19 683 | 638 | 42 | 19 655 | 335 | 48 | 11 755 |
| Arc-Flags | 1 220 | 74 | 1 067 | 1 588 | 80 | 710 | 1 276 | 83 | 1 038 | 670 | 96 | 618 |
| RE | 6 | 18 | 2 328 | 17 | 22 | 5 139 | 14 | 27 | 4 764 | 12 | 31 | 4 958 |
| uni REAL | 20 | 146 | 855 | 34 | 150 | 1 838 | 28 | 155 | 1 652 | 20 | 159 | 1 500 |
| REAL | 20 | 146 | 506 | 34 | 150 | 1 105 | 28 | 155 | 950 | 20 | 159 | 856 |
| HH | 2 | 45 | 660 | 4 | 115 | 679 | 4 | 128 | 677 | 4 | 207 | 661 |

The most interesting behavior is that of HH. On each subgraph the performance is almost the same as on the full graph. Recalling the way the hierarchy is built the reason is obvious. Preprocessing of HH starts with a contraction step of roughly building the 2-core and shortcutting degree-2 nodes. Thus, HH has no advantage when applying these steps before preprocessing.

### 4.3 Other Inputs

In order to gain further insights into the behavior of speed-up techniques, our last testsets use data that is completely different from road or railway networks. On the one hand, we test the performance of speed-up techniques in small world graphs and on the other hand, we want to evaluate the influence of density and diameter of the input on the performance of speed-up techniques. For our density testset we use unit-disc graphs used in the field of sensor networks (see [29] for a survey) with different average degrees. Our diameter testset uses multi-dimensional grid graphs with different numbers of dimensions as inputs.

**Small World.** Up to this point, we concentrated on graphs with some kind of hierarchy. In this test we use small world graphs as input without such a property. The first dataset represents the internet on the router level, i.e. nodes are routers and edges represent connections between routers. The network is taken from the CAIDA webpage [30] and has 190 914 nodes and 1 215 220 edges. The second graph is a citation network, i.e. nodes are papers and edges depict whether one paper cites another one. It is obtained from crawling the literature database DBLP [31] and has 268 495 nodes and 2 313 294 edges. The final dataset is a co-authorship [32] network (299 067 nodes and 1 955 352 edges) which is also obtained from the DBLP: Nodes represent authors and two authors are connected by an edge if they have written a paper together. The results for these data is shown in Tab. 6.

The most interesting observation is that the biggest speed-up is achieved by simply switching from uni- to bidirectional DIJKSTRA. This derives from the very small

**Table 6.** Performance of speed-up techniques on small world graphs.

| | router | | | | citations | | | | coAuthorship | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | |
| | min | B/n | #settled | ms | min | B/n | #settled | ms | min | B/n | #settled | ms |
| Dijkstra | 0 | 0 | 94 717 | 89.0 | 0 | 0 | 134 136 | 190.8 | 0 | 0 | 153 885 | 125.5 |
| BiDijkstra | 0 | 0 | 216 | 0.3 | 0 | 0 | 742 | 1.5 | 0 | 0 | 320 | 0.4 |
| uni ALT | 2 | 128 | 23 430 | 36.8 | 2 | 128 | 28 853 | 68.6 | 2 | 128 | 38 173 | 51.5 |
| ALT | 2 | 128 | 320 | 1.7 | 2 | 128 | 850 | 4.7 | 2 | 128 | 667 | 2.2 |
| uni Arc-F. | 351 | 102 | 5 453 | 12.9 | 1 488 | 138 | 46 318 | 113.7 | 507 | 105 | 28 225 | 62.8 |
| Arc-Flags | 702 | 204 | 42 | 0.1 | 2 977 | 276 | 231 | 0.7 | 1 014 | 209 | 117 | 0.3 |
| RE | 174 | 11 | 820 | 1.7 | 1 922 | 18 | 3 465 | 8.4 | 417 | 10 | 445 | 0.9 |
| uni REAL | 176 | 139 | 22 493 | 44.2 | 1 924 | 146 | 27 898 | 90.3 | 419 | 138 | 34 163 | 67.5 |
| REAL | 176 | 139 | 337 | 2.3 | 1 924 | 146 | 762 | 6.0 | 419 | 138 | 522 | 2.9 |
| HH | 38 | 1815 | 20 488 | 1 307.7 | 862 | 532 | 89 696 | 928.9 | 246 | 2982 | 61 703 | 1 713.7 |

diameter of the graph (less than 8 for all instances). Stunningly, only Arc-Flags yield an additional but only mild speed-up. Taking the huge preprocessing of more than 10 hours into account, the usage of Arc-Flags cannot be justified. Any other approach is even slower than bidirectional DIJKSTRA which is mainly due to computational overhead. Analyzing HH, this approach seems to have serious problems with small world graphs. The reason is the stopping criterion (cf. [20]). Normally, bidirectional search can be stopped as soon as both search spaces meet. But for HH, this does not hold: the search has to be continued as long as both searches have reached the highest core or when the forward search settles the target node.

We conclude that—as long as bidirectional search is allowed—no speed-up technique is applicable. However, the situtation changes if a scenario arises with small-world graphs and prohibited bidirectional search. In such a scenario, unidirectional ALT yields the best tradeoff between preprocessing time and query performance.

**Sensor Networks.** During the last years, the field of sensor networks has drawn wide attention. At a glance, routing in such networks has similar properties as routing in road networks. Thus, we evaluate so called unit disk graphs which are widely used for experimental evaluations [33] in that field. Such graphs are obtained by arranging nodes on the plane and connecting nodes with a distance below a given threshold. It is obvious that the density can be varied by applying different threshold values. In our setup, we use graphs with about 1 000 000 nodes and an average degree of 5, 7, and 10, respectively. As metric, we use the distance between nodes according to their embedding. The results can be found in Tab. 7.

Uni- and bidirectional DIJKSTRA settle roughly the same number of nodes independent of the average degree but query times again increase with higher density due to more relaxed edges. Analyzing ALT, the bidirectional variant is twice as fast as the unidirectional algorithm for the instance with degree 5 while for degree 10, both approaches are equal to each other with respect to query times. The decreasing search space of unidirectional ALT is due to the increasing number of edges. With more edges, the shortest path is very close to the flight distance between source and target. In such

**Table 7.** Performance of speed-up techniques on unit disk graphs with different average degree.

| | average deg. 5 | | | | average deg. 7 | | | | average deg. 10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | |
| | min | B/n | #settled | ms | min | B/n | #settled | ms | min | B/n | #settled | ms |
| Dijkstra | 0 | 0 | 487 818 | 257.3 | 0 | 0 | 521 874 | 330.1 | 0 | 0 | 502 683 | 399.0 |
| BiDijkstra | 0 | 0 | 299 077 | 164.4 | 0 | 0 | 340 801 | 225.1 | 0 | 0 | 325 803 | 269.4 |
| uni ALT | 8 | 128 | 22 476 | 17.1 | 8 | 128 | 16 634 | 15.1 | 10 | 128 | 14 561 | 16.0 |
| ALT | 8 | 128 | 9 222 | 8.5 | 8 | 128 | 10 565 | 11.8 | 10 | 128 | 11 749 | 15.6 |
| uni Arc-Flags | 53 | 80 | 8 556 | 7.9 | 299 | 112 | 16 445 | 16.8 | 801 | 160 | 21 413 | 24.2 |
| Arc-Flags | 105 | 160 | 2 091 | 1.8 | 598 | 224 | 4 761 | 4.6 | 1 602 | 320 | 7 019 | 7.5 |
| RE | 4 | 20 | 848 | 0.5 | 46 | 42 | 13 783 | 14.3 | 1 153 | 54 | 83 826 | 104.5 |
| uni REAL | 12 | 148 | 307 | 0.4 | 54 | 170 | 2 072 | 3.2 | 1 163 | 182 | 8 780 | 13.6 |
| REAL | 12 | 148 | 291 | 0.4 | 54 | 170 | 2 394 | 4.1 | 1 163 | 182 | 11 449 | 21.7 |
| HH | 2 | 251 | 203 | 0.2 | 12 | 549 | 5 068 | 8.5 | 71 | 690 | 23 756 | 49.1 |

instances, the potentials deriving from landmarks are very good. Arc-Flags yield very good query times but again for the price of high preprocessing times. Hierarchical methods work very good on average degrees of 5 and 7. For a degree of 10 preprocessing and query times increase drastically. For RE, a reason is that node-labels are used for pruning the search. With increasing density, many edges are never used by any shortest path. As these edges cannot be pruned by using node-labels, query times increase.

Summarizing, for low densities, hierarchical methods like HH/RE yield the best results on these instances, while ALT wins for high average degrees. Although Arc-Flags are faster with respect to query times, preprocessing is much faster for ALT.

**Grid Graphs.** Our last testset exploits the influence of graph diameter on the performance. Here, we vary the diameter of a graph by using multi-dimensional grid graphs with 2, 3, and 4 dimensions. The number of nodes is set to 250 000, and thus, the number of edges is 1, 1.5, and 2 million, respectively. Edge weights are picked uniformly at random from 1 to 1000. These results can be found in Tab. 8.

Like for sensor networks, unidirectional DIJKSTRA settles the same amount of nodes on all graphs. But due to more edges relaxed query times increase with an increasing number of dimensions. As the diameter shrinks with increasing an number of dimensions, bidirectional DIJKSTRA settles less nodes on 4-dimensional grids than 2-dimensional grids. We already observed this effect more drastically for small world graphs (cf. Tab. 6). This analysis also holds for the performance of uni- and bidirectional ALT. Our hierarchical representatives RE/HH perform very good on 2-dimensional grids but significantely lose performance when switching to higher dimensions. The main reason is that the contraction phase of the algorithms fail.

Summarizing, ALT has the best trade-off with respect to preprocessing and query times on higher-dimensional grids. Only Arc-Flags are faster but for the price of a much higher effort in preprocessing. Hierarchical methods like RE/HH can only compete with ALT on 2-dimensional grids.

**Table 8.** Performance of speed-up techniques on the grid graphs with different numbers of dimensions.

| | 2-dimensional | | | | 3-dimensional | | | | 4-dimensional | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PREPRO | | QUERY | | PREPRO | | QUERY | | PREPRO | | QUERY | |
| | min | B/n | #settled | ms | min | B/n | #settled | ms | min | B/n | #settled | ms |
| Dijkstra | 0 | 0 | 125 675 | 36.7 | 0 | 0 | 125 398 | 78.6 | 0 | 0 | 122 796 | 137.5 |
| BiDijkstra | 0 | 0 | 79 962 | 24.2 | 0 | 0 | 45 269 | 28.2 | 0 | 0 | 21 763 | 20.3 |
| uni ALT | 1 | 128 | 5 452 | 2.5 | 2 | 128 | 4 223 | 3.8 | 3 | 128 | 5 031 | 7.5 |
| ALT | 1 | 128 | 2 381 | 1.5 | 2 | 128 | 1 807 | 2.2 | 3 | 128 | 1 329 | 2.5 |
| uni Arc-Flags | 45 | 64 | 4 476 | 1.9 | 415 | 94 | 8 996 | 5.7 | 1 559 | 122 | 25 125 | 26.8 |
| Arc-Flags | 89 | 128 | 1 340 | 0.6 | 830 | 189 | 1 685 | 1.0 | 3 117 | 244 | 2 800 | 2.3 |
| RE | 13 | 31 | 3 797 | 2.1 | 220 | 102 | 18 177 | 27.1 | 2 243 | 89 | 20 587 | 40.2 |
| uni REAL | 14 | 159 | 799 | 0.8 | 222 | 230 | 5 081 | 10.6 | 2 246 | 217 | 10 740 | 30.3 |
| REAL | 14 | 159 | 829 | 0.9 | 222 | 230 | 3 325 | 8.5 | 2 246 | 217 | 3 250 | 11.6 |
| HH | 2 | 1682 | 583 | 0.6 | 32 | 1954 | 17 243 | 95.8 | 680 | 662 | 61 715 | 343.0 |

## 5 Conclusion and Outlook

We learned a lot about the performance of the most prominent speed-up techniques on graph classes other than road networks. For timetable information, the speed-up achieved on time-expanded graphs is much smaller than the speed-up on road network, even without necessary modifications that will most probably decrease performance. Even worse, the speed-up obtained by all techniques is below the blow-up factor of approximately 250 between time-dependent and corresponding time-expanded graphs. We observed that plain DIJKSTRA yields lower query times on a condensed network than any other speed-up techniques on the time-expanded graphs. Recall that the time-dependent model can be interpreted as an extension of the condensed one. In [27], it is shown that plain DIJKSTRA can be used in a dynamic time-dependent scenario easily, and time-dependent ALT achieves an additional speed-up of factor 5 over plain DIJKSTRA [16]. In addition, incorporating delays seems to be easier in the time-dependent model than in the time-expanded one. We conclude that it is promising to work on the dynamic time-dependent model for solving the timetable information problem.

Regarding time-expanded data, we do not see an alternative to real-world data: on other inputs, all examined speed-up techniques perform completely different than on our real-world time-expanded datasets. However, road networks seem to be a good alternative for condensed graphs and thus, also for the time-dependent model. We expect that an approach working well in a (dynamic) time-dependent road network will also perform well on (dynamic) time-dependent railway networks.

Concerning speed-up techniques in general, we gained further and interesting insights by our extensive experimental study. Hierarchical approaches seem to have problems with high-density networks, the chosen metric has a high impact on achieved speed-ups, edge-labels are somewhat superior to node-labels, and small diameters yield big speed-ups for bidirectional search. As a consequence, the choice of which technique to use highly depends on the scenario. However, of all examined speed-up techniques, ALT provides a reasonable trade-off of preprocessing time and space on the one hand and achieved speed-up on the other hand. Although this approach is slower on hierarchical inputs it is more *robust* with respect to the input. In addition, ALT works in dynamic and time-dependent scenarios.

We see a lot of future work for speed-up techniques on timetable information systems. First of all, we plan to tackle the dynamic time-dependent approach. However, as soon as we do multicriteria routing, e.g. minimize number of transfers, the time-expanded model has several advantages over the time-dependent one [7]. Thus, it seems promising to develop new speed-up techniques tailored for the time-expanded model that exploit specific properties of these graphs. We assume that such highly specialized techniques can compete with the time-dependent approach. However, the problem of incorporating delays in expanded graphs persists.

**Acknowledgments.** We would like to thank Dominik Schultes for providing the Highway Hierarchies code and his help on parameter settings. We also thank Robert Görke and Bastian Katz for providing data and Daniel Karch for implementing Arc-Flags.

## References


1. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
2. Wagner, D., Willhalm, T.: Speed-Up Techniques for Shortest-Path Computations. In: 24th International Symposium on Theoretical Aspects of Computer Science (STACS). (2007) 23–36
3. Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 23–36
4. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Graphs. In: 9th DIMACS Challenge on Shortest Paths. (2006)
5. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Time Shortest-Path Queries in Road Networks. In: Algorithm Engineering and Experiments (ALENEX). (2007) 46–59
6. 9th DIMACS Implementation Challenge: Shortest Paths. `http://www.dis.uniroma1.it/~challenge9/` (2006)
7. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. In et. al., F.G., ed.: Algorithmic Methods for Railway Optimization. Volume 4359 of Lecture Notes in Computer Science., Springer Verlag (2007) 67–90
8. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining speed-up techniques for shortest-path computations. ACM Journal of Experimental Algorithmics **10** (2005) article 2.5
9. Goldberg, A., Kaplan, H., Werneck, R.: Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In: Algorithm Engineering and Experiments (ALENEX). (2006) 129–143
10. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. ACM Journal of Experimental Algorithmics **12** (2007) article 2.4
11. Schulz, F., Wagner, D., Zaroliagis, C.: Using multi-level graphs for timetable information in railway systems. In: Proc. Algorithm Engineering and Experiments. Volume 2409 of LNCS., Springer (2002) 43–59
12. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric containers for efficient shortest-path computation. ACM Journal of Experimental Algorithmics **10** (2005) 1–30
13. Goldberg, A.V., Harrelson, C.: Computing the shortest path: $A^*$ meets graph theory. In: 16th ACM-SIAM Symposium on Discrete Algorithms. (2005) 156–165
14. Hart, P.J., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4** (1968) 100–107
15. Goldberg, A.V., Werneck, R.F.: An efficient external memory shortest path algorithm. In: Algorithm Engineering and Experimentation (ALENEX). (2005) 26–40
16. Delling, D., Wagner, D.: Landmark-Based Routing in Dynamic Graphs. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 52–65
17. Lauther, U.: An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In: Geoinformation und Mobilität – von der Forschung zur praktischen Anwendung. Volume 22., IfGI prints, Institut für Geoinformatik, Münster (2004) 219–230
18. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speed up Dijkstra's algorithm. In: 4th International Workshop on Efficient and Experimental Algorithms. (2005) 189–202
19. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computation with Arc-Flags. In: 9th DIMACS Challenge on Shortest Paths. (2006)

20. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms (ESA). Volume 4168 of LNCS., Springer (2006) 804–816
21. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In: 9th DIMACS Challenge on Shortest Paths. (2006)
22. Gutman, R.J.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Algorithm Engineering and Experiments (ALENEX), SIAM (2004) 100–111
23. HaCon: Ingenieurgesellschaft mbH. `http://www.hacon.de` (1984)
24. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03). Volume 92 of Electronic Notes in Theoretical Computer Science., Elsevier (2004) 85–103
25. METIS: A family of multilevel partinioning algorithms. `http://glaros.dtc.umn.edu/gkhome/views/metis/` (1995)
26. Goldberg, A.V., Kaplan, H., Werneck, R.: Better Landmarks within Reach. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 38–51
27. Cooke, K., Halsey, E.: The shortest route through a network with time-dependent intemodal transit times. Journal of Mathematical Analysis and Applications **14** (1966) 493–498
28. PTV AG: Planung Transport Verkehr. `http://www.ptv.de` (1979)
29. Rajaraman, R.: Topology Control and Routing in Ad hoc Networks: A Survey. SIGACT News **33** (2002) 60–73
30. CAIDA: Cooperative Association for Internet Data Analysis. `http://www.caida.org/` (2001)
31. DBLP: DataBase systems and Logic Programming. `http://dblp.uni-trier.de/` (2007)
32. An, Y., Janssen, J., Milios, E.E.: Characterizing and mining the citation graph of the computer science literature. Knowl. Inf. Syst. **6** (2004) 664–678
33. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'03). (2003)